Dr.D.Satyanarayana, 2026, 14:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

High speed floating point multiplication on FPGA using carry-look ahead adders

[1]Dr.D.Satyanarayana,[2]Dr.A.Ranganayakulu, [3]pagadala Himanjali,[4]Thota Lavanya,[5]Sadu Snehalatha,[6]Kappeta Madhavi

[1]Assosiate professor,Department of ECE,KITS-Markapur
[2]Professor&Head,Department of ECE,KITS-Markapur
[3,4,5,6] Students,Department of ECE,KITS-Markapur

Abstract—Multipliers are crucial components for implementing algorithms for processing digital signals in hardware. When planning the architecture of a system as a whole, multiplier design is crucial. The floating-point format is preferred over the fixed-point representation for algorithms that need data with a dynamic range. However, designers have challenges with floating-point multipliers because to their enormous latency and space requirements. In this paper, we propose a spatially and temporally improved approach to floating point multiplication. The Mantissa multiplication, which is carried out by rapid tree multipliers, is the design bottleneck of the floating-point multiplier. The suggested architecture improved partial product reduction in mantissa multiplication by using Carry-lookahead adders as compressors, as opposed to half-adders and full-adders used in traditional Wallace or Dadda tree multipliers. Verilog HDL description is used to develop the FPGA and verify the suggested design. There was a 9.9 percentage point reduction in latency compared to Dadda and an 8.5% improvement compared to Wallace.

Index Terms—Floating-point, FPGA, Tree multipliers.

I. INTRODUCTION

The last step of adding produces the desired outcome. The most time-consuming part of the calculation is the reduction step. To improve reduction efficiency, other approaches have been suggested, such as Dadda Tree multipliers [15] and Wallace Tree multipliers [14]. When it comes time to reduce, both approaches utilize full and half adders. The greatest reduction ratio that has been achieved is 3:1. It means that in the reduction phase, three bits are reduced to two bits by sending partial products to a single stage of complete adders. One may foresee and complete the incomplete products produced by booth coding by using the symbolic expansion notion, as shown in [16] with the help of the enlarged booth wallace approach. The partial products obtained by adding them together and by using single precision floating-point multiplication are also amenable to optimization. In [17], the authors suggest units that use a combined Wallace and Dadda structure to efficiently multiply mantissas.

As the second most common operation, after addition, in high performance computing structures like filter design [1], Multiplier Design is an important consideration. Sometimes a processing job's success or failure is determined by how quickly a multiplication can be performed [2]. The data type that should be used for multiplier design is determined by the applications' requirements for accuracy and dynamic range. Using floating-point format expands the range of possible values without increasing the bit-width. It also sidesteps the overflow and underflow problems that plague fixed-point digital architectures [3]. The IEEE-754 standard specifies two main categories for floating point formats: 32-bit single precision and 64-bit double precision. For most uses, a 32-bit format provides more than enough room for the range of values that are required. Due to this, the 32-bit floating point multiplier method and its related issues were investigated, and it was found

that the mantissa multiplication unit had the lowest performance [4]-[13]. There are typically three steps involved when multiplying two binary values. The first step is to generate a partial product using logical and operational processes. Additionally, this study demonstrates that carry-lookahead adders may achieve a greater reduction ratio of 9:5, which is great news for partial products. In the reduction step, the suggested approach shortens the time it takes. Using carry lookahead adders will increase overall performance since 24-bits by 24-bits mantissa multiplication is quite expensive in floating-point multiplication.

II. FLOATING-POINT MULTIPLICATION

Various floating-point formats may stand in for distinct subsets of floating-point data based on their radix, precision, and exponent range, as per the latest IEEE standard for floating-point arithmetic. (−1)sign × bexponent × mantissa is the formula for floating-point numbers, where radix is 2 for binary and 10 for decimal, and precision is the number of digits in the mantissa [18]. The sign bit (S) in the 32-bit binary format shows whether the number is positive or negative. The biased exponent (E = e + bias), where e is the real exponent and the bias value is 127, is represented by the 8 bits that follow. The mantissa is the remaining 23 bits. As seen in Figure 1, this is the floating-point format. The following steps may be taken to elaborate the floating-point multiplication: 1) The exclusive-or operation is used to acquire the sign bit of the product by passing the sign bits of the multiplier and multiplicand through it.

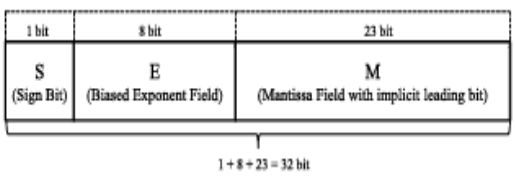

Image 1: 32-bit floating-point representation 2) The multiplier and the multipli cand are multiplied by their 8-bit exponents. Third, double the mantissa of the multiplier by the multiplicand. 4) The final product exponent is updated by first performing Mantissa multiplication, and then rounding and normalizing the results.

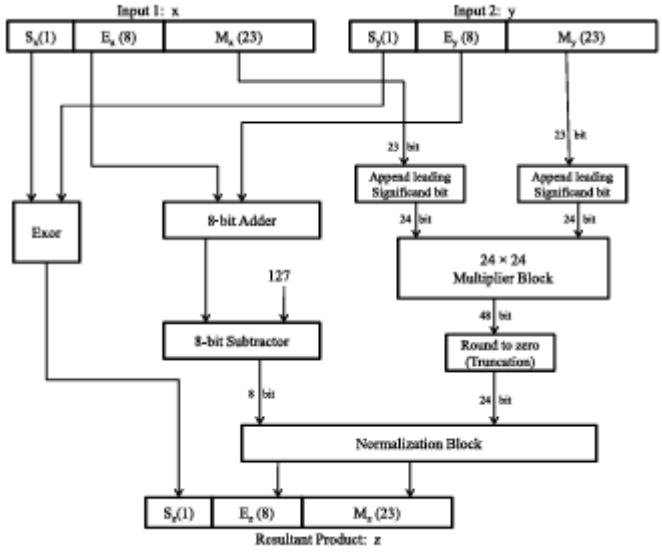

Fig. 2. 32 bit floating-point multiplication

Figure 2 shows a block diagram of the processes described earlier.

It is possible to execute many operations in parallel, including those that calculate the sign bit, add exponents, and multiply mantissas. Before normalization and rounding can be done, however, the product from the mantissa multiplication has to be evaluated. Most digital signal processing tasks are well-suited to rounding to the nearest integer. Therefore, we consider this in our method by keeping the upper 24 bits of the 48-bit product generated by the 24 × 24 mantissa multiplication and releasing the lower 24 bits.You may change the exponent by locating the product's leading 1 and adding or removing it. According to the identified leading position, the radix point will be moved to the left when the exponent is increased and to the right when the exponent is decreased.

III. MANTISSA MULTIPLICATION

The mantissa multiplication is performed using an unsigned 24-bit multiplier. It is divided into three parts. The first step in creating partial products is to use the logical AND operator. Phase two involves zeroing down the partial product matrix to just two rows: the sum row and the carry row. To get the product, these two rows are added together. Time is primarily wasted on partial-product reduction, which often necessitates tree architectures with parallel adders. Once the reduction tree has shrunk the 24 × 24 matrix to only two rows, the final result may be computed using the appropriate ripple carry adder (RCA). To reduce the partial products, the current approaches use full adders and half adders. Dot notation, as shown in Figure 3, allows for a more clear visualization of such tree multipliers. Bit locations, not bit values, are what matter in dot representation, which is why dots are used to denote bit positions.

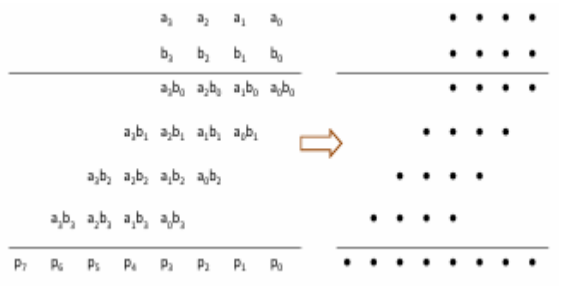

Fig. 3. Dot Diagram

We may describe the Wallace tree multiplication method as follows: 1) Once the partial product array is generated, adjacent rows are grouped into non-overlapping sets of three. 2) You may reduce the number of rows in each pair of three by. a) A full-adder is used for each column that has three bits. b) For every two-bit column, a half-adder is used. c) The following step is skipped for each column that has a single bit. 3) The reduction approach is repeated in each succeeding step until only two rows remain. 4) The final two rows are enhanced using a Carry Propogation Adder. For 4-by-4 multiplication, the aforementioned procedure is shown in Fig. 4. To make multiplication go more quickly, you may either add the operands (partial products) more quickly or decrease the number of them. For mantissa multiplication, there is a hard limit of 24 rows of partial products for the number of operands. This paper suggests using carry look-ahead adders to expedite the reduction of partial products. Figure 5 depicts a 4-bit carry look-ahead adder (CLA4) that can accept up to 9 PP bits as input and output no more than 5 PP bits, a ratio of 1.8 to 1. A multiplier that requires fewer reduction steps than Wallace/Dadda may be constructed using CLA4s since they are faster at PP reduction than complete adders. A CLA's reduction stage fails if the input size is more than 4 bits.

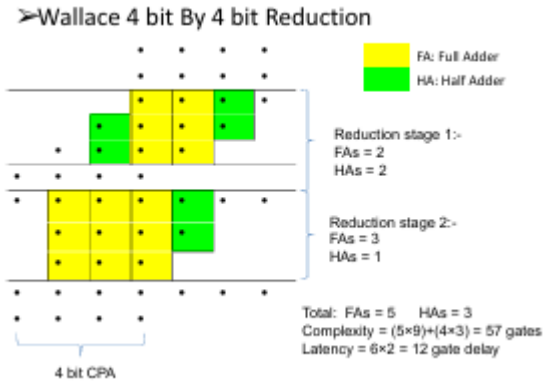

Fig.4. Wallace4bitBy4bitReduction

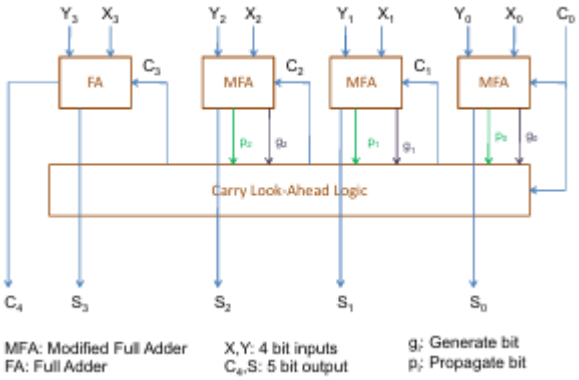

Fig.5. 4bitcarrylookaheadadder performancebeginstodegrade.

This is so because there is a clear correlation between delays in CLAs with input sizes greater than four and delays in gate delays.The advantage of raising the input size is entirely rendered null and void by this.The reduction of an

8×8 partial product tree is shown in Figure 6.The number of reduction steps for an 8×8 multiplier is 4 in the case of wallace/dadda, however for a structure based on CLA4, the reduction is achieved in 3 stages. When compared to the Wallace/Dadda multiplier, the CLA4-based multiplier will be quicker since there are fewer reduction steps.

To get the best results, use a high concentration of CLA4s since they have a higher reduction ratio.But there were problems with CLAs as well.The CLA multiplier takes on a less regular structure after the first phase of simplification. Using a negotiated technique to put CLA4s is unsuccessful in several contexts. Depending on the position of a certain CLA4, an extra delay step may be necessary.This indicates that CLA4 is not regularly inserted after the first step.

The parallelism in the CLA4 structure results in a latency of six gate delays, which is comparable to the latency of a complete adder.But compared to complete ladders, CLA4's difficulty is rather high. The comparison of gate delays for various multiplier sizes is shown in Figure 7. Using carrylook-ahead adders in the reduction phase decreases the number of stages and, by extension, the total latency, as is seen in the previous comparison. The quantity of components used for is shown in Table I.

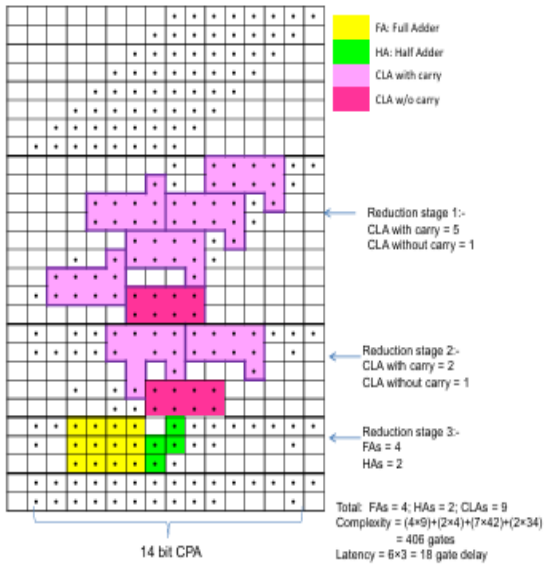

Fig.6. 8bitby8bit reductionusingCLAs

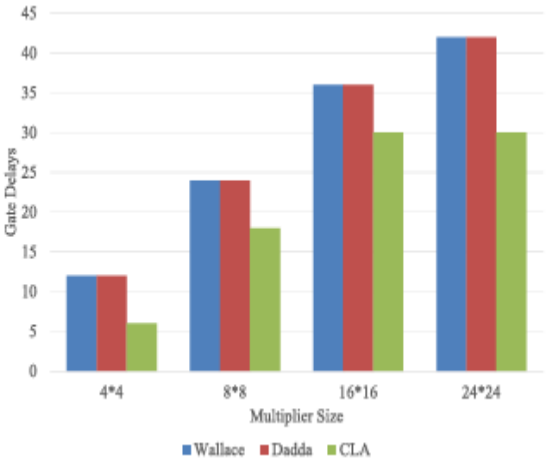

Fig.7. LatencyComparison designingmultipliersofdifferent sizes.

TABLEI UNITUTILIZATIONCLABASEDMULTIPLIER

Units used	Size		
	*8*8*	*16*16*	*24*24*
Carry Look-ahead Adder	9	43	131
Full Adder	4	27	29
Half Adder	2	39	22

IV. FPGA IMPLEMENTATION

In recent years, the field programmable gate array (FPGA) has emerged as a leading platform for

digital system prototyping.Digital systems may be described and modeled using hardware description languages (HDLs) like Verilog and VHDL in order to target a certain field-programmable gate array (FPGA) device. A floating point multiplier design model is created in this research using the Verilog Hardware Descrip tion Language (HDL). Then, using Xilinx Vivado 14.7 and the Artix 7 Field-Programmable Gate Array (FPGA) as our target, we synthesise the HDL representation into generic gate-level components. Figure 8 displays the waveform of the test-bench simulation. The result of multiplying the two integers 4.125 and 3.707 is 15.291375.

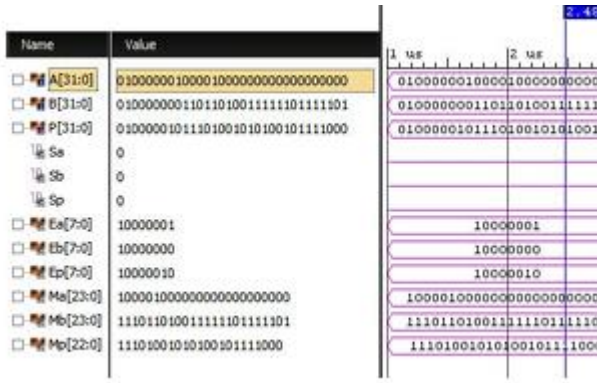

Fig. 8. Simulation Output Waveforms

The implementation report shown in Table II was generated after behavioral modeling, synthesis, and implementation. It was also necessary to include Verilog descriptions for floating-point multipliers using Wallace trees and Dadda tree based Mantissa multiplication in order to evaluate the suggested structure in comparison to the current traditional methods. Overall latency will be reduced as a result of the reported reduction in path delay.

TABLE II FPGA IMPLEMENTATION REPORT

Mantissa Multiplier Using	Max. Path Delay	Device Utilization
Wallace Tree	36.53ns	852
Dadda Tree	37.13ns	857
CLA Tree (Proposed)	33.42ns	925

V. CONCLUSION

In terms of hardware implementation, this paper suggests a way to do floating point multiplication that is more efficient for digital signal processing applications. In traditional Dadda or Wallace tree multipliers, Carry-lookahead adders are used as compressors rather than half adders and full adders. Because of this, the latency is reduced during mantissa multiplication, and the partial product reduction is improved. An 8.5% improvement compared to Wallace and a 9.9% improvement compared to Dadda in route latency were achieved by using Verilog. As a trade-off for delay, however, device consumption rose. When dealing with data ranges that are subject to change, this method works better.

REFERENCES

1. M. Gupta, M. Kansal, S. Thyagarajan, P.S. Chauhan,D.K. Upad hyay,"Design and Analysis of Non-uniform Transmission Line Based Dual-Band Bandpass Filter," Advances in VLSI, Communication, and Signal Processing. Lecture Notes in Electrical Engineering, vol 911, Springer, Singapore, 2022, pp. 507–518.
2. S. Abraham, S. Kaur and S. Singh, "Study of various high speed mul tipliers," 2015 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2015, pp. 1-5.
3. C. Inacio and D. Ombres, "The DSP decision: fixed point or floating?,"in IEEE Spectrum, vol. 33, no. 9, pp. 72-74, Sept. 1996.
4. D. Goldberg, "What every computer scientist should know about f loating-point arithmetic." ACM computing surveys (CSUR) vol. 23, no. 1, pp. 5-48, 1991.
5. J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," 2013 18th IEEE European Test Symposium (ETS), Avignon, pp. 1-6,27-30 May 2013.

6. H. Jiang, C. Liu, N. Maheshwari, F. Lombardi and J. Han, "A comparative evaluation of approximate multipliers," 2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Beijing, pp. 191-196, 18-20 July 2016.
7. IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2008 , vol., no., pp.1-70, Aug. 29 2008. Behrooz Prahami, Computer Arithmetic Algorithms and Hardware De signs, Oxford.
8. K. V. Gowreesrinivas and P. Samundiswary, "Comparative analysis of single precision floating point multiplication using compressor tech niques," IEEE International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, pp. 2428 2433, 22-24 March 2017.
9. M. Mehta, V. Parmar, E. Swartzlander, "High-speed multiplier design us ing multi-input counter and compressor circuits", Computer Arithmetic 1991. Proceedings. 10th IEEE Symposium on, pp. 43-50, Jun 1991.
10. S. Asif and Y. Kong, "Design of an algorithmic Wallace multiplier using high speed counters," International Conference on Computer Engineering and Systems (ICCES), Cairo, pp. 133-138,23-24 Dec. 2015.
11. Abhay Sharma,"FPGA Implementation of a High Speed Multiplier Employing Carry Lookahead Adders in Reduction Phase", International Journal of Computer Applications vol 116 number,17 pp. 27-31, April 2015.
12. K. Arun and K. Srivatsan, "A binary high speed floating point mul tiplier," IEEE International Conference on Nextgen Electronic Tech nologies: Silicon to Software (ICNETS2), Chennai, pp. 316-321, 23-25 March 2017.
13. C.S. Wallace, "A suggestion for fast multiplier," IEEE trans. Electron. Comput., vol. EC-13, pp. 14-17, 1964.
14. L. Dadda, "Some schemes for parallel multipliers", Alta Frequenza, vol. 34, pp. 349-356, 1965.
15. Na Bai, Hang Li, Jiming Lv, Shuai Yang, Yaohua Xu, "Logic Design and Power Optimization of Floating-Point Multipliers", Computational Intelligence and Neuroscience, vol. 2022, Article ID 6949846, 2022.
16. Anuradha, Sujit Kumar Patel, Subodh Kumar Singhal, "An area-delay efficient single-precision floating-point multiplier for VLSI systems", Microprocessors and Microsystems, Vol. 98, 2023.
17. IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2008 , vol., no., pp.1-70, Aug. 29 2008